A Balanced Introduction to Computer Science

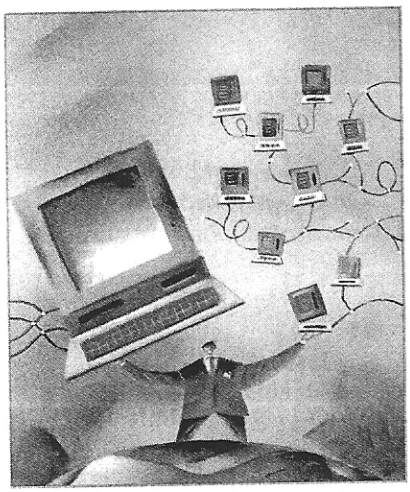
Second Edition

David Reed

Creighton University

Buttons & Text boxes
pp. 115 - 124

Upper Saddle River, NJ 07468



Event-Driven Pages

Chapter 7

This chapter uses narrative, examples, and hands-on exercises to introduce programming concepts and Web development skills.

The Internet already has too many static pages. Why add yours to the list? To get noticed, your page must be live, it must do something, or offer some service. What better way is there to liven pages up yourself than to use JavaScript, writing a short script that runs in the browser, right in front of the user? JavaScript puts all the elements in a web page—and the web page itself!—under your control.

Steven Holzner
Essential JavaScript

When I can't handle events, I let them handle themselves.

Henry Ford

Chapter Outline

- Initiating Actions via Buttons
- Input/Output via Text Boxes
  - Text Boxes for Displaying Output
  - Text Boxes for Accessing Input
  - Text Boxes for Handling Both Input and Output
- Input/Output via Text Areas
- Dynamic Images
- Simplifying Buttons with Functions
- Looking Ahead...
- Chapter Summary

One feature that makes the World Wide Web so popular is its interactive nature. When you access a Web page, you don't just view it—you interact with it. You click links and buttons to access other pages or to make new windows appear, or you enter information and view responses based on your input. In these and many other ways, Web pages respond to your actions. We refer to such Web pages as "event-driven," in that they react to user-initiated events such as mouse clicks or keyboard entries.

Programmers can use JavaScript code to specify the way in which Web pages handle various events. Thus, you can now apply your JavaScript programming skills to writing event-driven Web pages. For example, instead of prompting users to enter a list of grades, you can design a page in which users type grades in boxes and then click a button to compute their course average. Although the same JavaScript code can compute the average in either case, the event-driven page offers the user a more attractive and easy-to-use interface.

In this chapter, you will begin using JavaScript as a tool for controlling event-driven Web pages. The first section demonstrates the use of buttons to initiate actions. We then introduce text boxes and text areas, HTML elements used to read in user input and display output. At the end of the chapter, you will learn how to define simple functions to more easily control event-driven pages. It is important to note that the event-driven features introduced in this chapter are far from exhaustive. In fact, HTML provides numerous additional elements (such as radio buttons, check boxes, and selections lists) and JavaScript provides

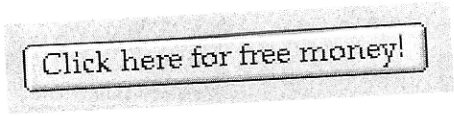


Figure 7.1 An HTML button.

additional methods for controlling those elements. However, buttons, text boxes, and text areas are the most commonly used tools for building event-driven pages and in many situations are sufficient. This book will focus exclusively on these page elements.

Initiating Actions via Buttons

An ***event handler*** is an HTML element that can be programmed (using JavaScript) to respond to a user's actions. The simplest type of event handler that can appear in Web pages is a button. If you have spent any time surfing the Web, you have no doubt run across buttons like the one shown in Figure 7.1.

A ***button*** is an HTML element that you can embed in a Web page, just as you would an image or a table. Unlike these static elements, however, buttons can be associated with JavaScript code that will run only when the buttons are clicked. For example, the button in Figure 7.1 might be assigned JavaScript statements that cause a new page or window to open. When a user clicks this button, the associated code will execute, redirecting the user's browser to the specified page. The following represents a generalized button element:

```
<input type="button" value="BUTTON_LABEL" onclick="JAVASCRIPT_CODE" />
```

A button's VALUE attribute specifies the text label that appears on the button. For example, the button in Figure 7.1 has the following attribute assignment: `value="Click here for free money!"`. The ONCLICK attribute specifies the action that occurs when a user clicks the button. This attribute can be assigned any JavaScript statement or sequence of statements, enclosed in quotes.

The Web page in Figure 7.2 contains a button with the label `"Click Here for Lucky Number"`. Once this page has been loaded, clicking the button will execute the two JavaScript statements

```
1.  <html>                                                    Dave Reed -->
2.    <!-- lucky.html                                                    -->
3.    <!-- This page displays a lucky number at the click of a button.  -->
4.    <!-- ============================================================== -->
5.
6.    <head>
7.      <title> Dave's Lucky Number </title>
8.    </head>
9.
10.   <body>
11.     <div style="text-align:center">
12.       <h2>Lucky Dave's Gift To You</h2>
13.
14.       <p>Numbers rule our lives. If you would like the benefit of Lucky Dave's
15.       amazing powers of prognostication, click on the button below to receive
16.       your guaranteed lucky number for the day!</p>
17.
18.       <input type="button" value="Click Here for Lucky Number"
19.              onclick="luckyNum = Math.floor(Math.random()*100)+1;
20.                       alert('Your lucky number is ' + luckyNum);" />
21.
22.     </div>
23.   </body>
    </html>
```

Figure 7.2 Simple Web page containing a button.

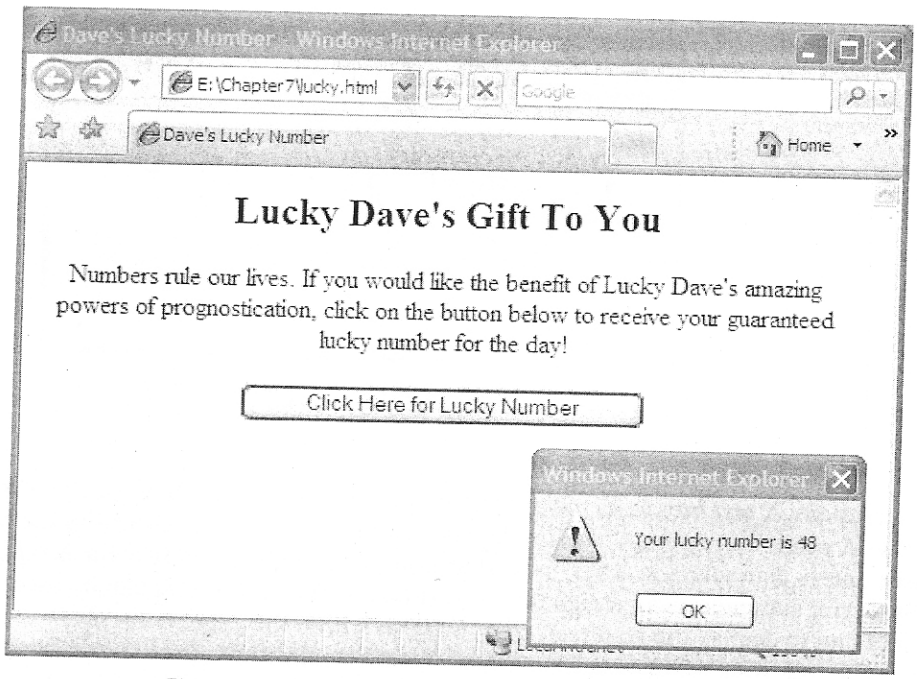


Figure 7.3 `lucky.html` rendered in a Web browser.

contained in the ONCLICK attribute. The first statement generates a random lucky number from the range 1 to 100, and assigns that number to the variable `luckyNum`. The second statement displays the lucky number in an alert window (Figure 7.3).

You should note that the second statement in the ONCLICK attribute uses single quotes to specify the alert message to be displayed. As was mentioned in the supplemental section of Chapter 2, a JavaScript string can be specified using either double quotes ("...") or single quotes ('...'). Because the alert statement is enclosed within the double quotes of the ONCLICK attribute, it must use single quotes for the message to avoid a conflict. Otherwise, the browser would be confused by the nested double quotes and would fail to define the attribute.

EXERCISE 7.1

Enter the contents of `lucky.html` from Figure 7.2 into a new Web page, then load the page in the browser to verify that it behaves as described.

Once you have done this, save a copy of the page under the name `lotto.html` and modify it so that it predicts the winning combination for a PICK-4 lottery. You should update the page's text appropriately and modify the button so that its label reads `"Click for Today's PICK-4 Winner"`. Instead of displaying a single number, the JavaScript code associated with the button should select four random numbers from the range 0 to 9 and then display them in an alert window, separated by dashes. Figure 7.4 depicts an example of an alert window that contains a PICK-4 winning number.

Input/Output via Text Boxes

A button provides a simple mechanism through which users can interact with a Web page. By clicking the button, the user initiates some action. Although an alert window is sufficient for displaying a simple message, most tasks require more elaborate types of user interaction, such as where the user enters several inputs and then views the results of a computation. Fortunately, HTML provides another event-handler element, called a text box, that can be used to implement these more complex

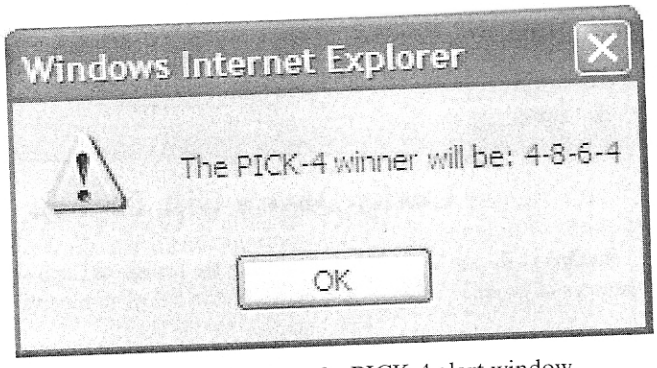


Figure 7.4 Sample of a PICK-4 alert window.

interactions. A ***text box***, as its name suggests, is a box that can contain text (words or phrases). Unlike an alert window or prompt window, however, a text box is displayed within the page itself. When the user enters characters in a text box, that input is directly accessible via JavaScript code; similarly, JavaScript statements can assign text to text boxes. Thus, a text box can be used both to receive user input and to display the results of computations within the page.

The following represents a generalized text box element:

```
<input type="text" id="TEXTBOX_ID" size= "NUM_CHARS" value="INITIAL_TEXT" />
```

A text box's ID attribute specifies the name used to identify that box. The SIZE attribute defines the size of the box, which is measured by the number of characters it can contain. The VALUE attribute (if present) specifies the text that the text box initially contains.

As you contemplate the functionality of text boxes, it may help you to think of memory cells as an analogy. Recall that every JavaScript variable is associated with a specific memory cell—each reference to a variable accesses the value in its memory cell, and each assignment to the variable stores a value in that same memory cell (overwriting any value that was previously saved there). As you will see in the following sections, a text box stores a value that can be accessed and assigned in a similar manner. The main difference between a variable and a text box is that the contents of a text box are visible in the page, whereas the value of a variable is hidden.

Text Boxes for Displaying Output

Using JavaScript code, programmers can display a message in a text box by encapsulating the message as a string and assigning the string to the box's VALUE attribute. When you assign a value to a text box, you must specify the box by its ID: this is accomplished by typing the word `document`, followed by a period and the function call `getElementById('TEXTBOX_ID')`, where `TEXTBOX_ID` is the name assigned to the ID attribute of the text box. The following represents a generalized assignment to a text box:

```
document.getElementById('TEXTBOX_ID').value = STRING_TO_BE_DISPLAYED;
```

For example, Figure 7.5 depicts a modified version of the `lucky.html` page, which uses a text box, rather than a separate alert window, to display the lucky number. The text box, named `numberBox`, is four characters wide and initially appears blank (i.e., its initial contents consist of an empty string `""`). When the user clicks the button labeled `"Click Here for Lucky Number"`, a random lucky number between 1 and 100 is picked, as in Figure 7.3. However, instead of displaying the number in an alert window, the function displays the number in the text box via the assignment: `document.getElementById('numberBox').value = luckyNum`. The rendered page is shown in Figure 7.6.

EXERCISE 7.2

Enter the revised `lucky.html` text from Figure 7.5 into a Web page, then load the page in the browser to verify that it behaves as described.

```
<html>
  <!-- lucky1.html                                          Dave Reed -->
  <!-- This page displays a lucky number at the click of a button. -->
  <!-- ================================================================ -->

  <head>
    <title> Dave's Lucky Number </title>
  </head>

  <body>
   <div style="text-align:center">
     <h2>Lucky Dave's Gift To You</h2>

     <p>Numbers rule our lives. If you would like the benefit of Lucky Dave's
     amazing powers of prognostication, click on the button below to receive
     your guaranteed lucky number for the day!</p>

     <input type="button" value="Click Here for Lucky Number"
            onclick="luckyNum = Math.floor(Math.random()*100) +1;
                     document.getElementById('numberBox').value = luckyNum;" />
     <hr />
     <p>
     Your lucky number is: <input type="text" id="numberBox" size="4" value="" />
     </p>
   </div>
  </body>
</html>
```

Figure 7.5 Web page that displays a number in a text box.

Figure 7.6 `lucky1.html` rendered in a Web browser.

Make similar modifications to your lotto.html page, causing it to display the winning PICK-4 lottery numbers in a text box. This means that the JavaScript statements associated with your button must generate the four random numbers, concatenate them so that they are separated by hyphens, and assign the resulting string to a text box.

Common errors to avoid...

Like variables, text boxes must have unique identifiers associated with them. If you mistakenly use the same identifier to represent multiple text boxes, no error will occur. However, the identifier will be associated with the last text box that you defined, and all subsequent accesses using that identifier will refer to that last box.

Text Boxes for Accessing Input

Whereas some text boxes are used to display output, others are employed to handle user input. When a text box is intended to accept input, users can enter data by clicking the mouse pointer inside the box and typing. JavaScript code then accesses the text box's contents using the text box's identifier (where TEXTBOX_ID will vary for each text box):

```
document.getElementById('TEXTBOX_ID').value
```

For example, the Web page in Figure 7.7 is an event-driven version of the temperature conversion page `ftoc.html` from Chapter 7. The page contains a text box named `fahrBox`, which accepts

```
1. <html>
2.   <!-- convert.html                                          Dave Reed -->
3.   <!-- This page converts temperatures from Fahrenheit to Celsius. -->
4.   <!-- ================================================================ -->
5.
6.   <head>
7.     <title>Temperature Conversion Page</title>
8.   </head>
9.
10.  <body>
11.    <h2>Temperature Conversion Page</h2>
12.    <hr />
13.    <p>
14.      Enter a temperature in degrees Fahrenheit:
15.      <input type="text" id="fahrBox" size="10" value="" />
16.    </p>
17.    <p>
18.      <input type="button" value="Convert to Celsius"
19.             onclick="tempInF = document.getElementById('fahrBox').value;
20.                      tempInF = parseFloat(tempInF);
21.                      tempInC = (5/9) * (tempInF - 32);
22.                      document.getElementById('celsiusBox').value = tempInC;" />
23.    </p>
24.    <p>
25.      Equivalent temperature in degrees Celsius:
26.      <input type="text" id="celsiusBox" size="10" value="" />
27.    </p>
28.  </body>
29. </html>
```

Figure 7.7 Web page that uses text boxes for input and output.

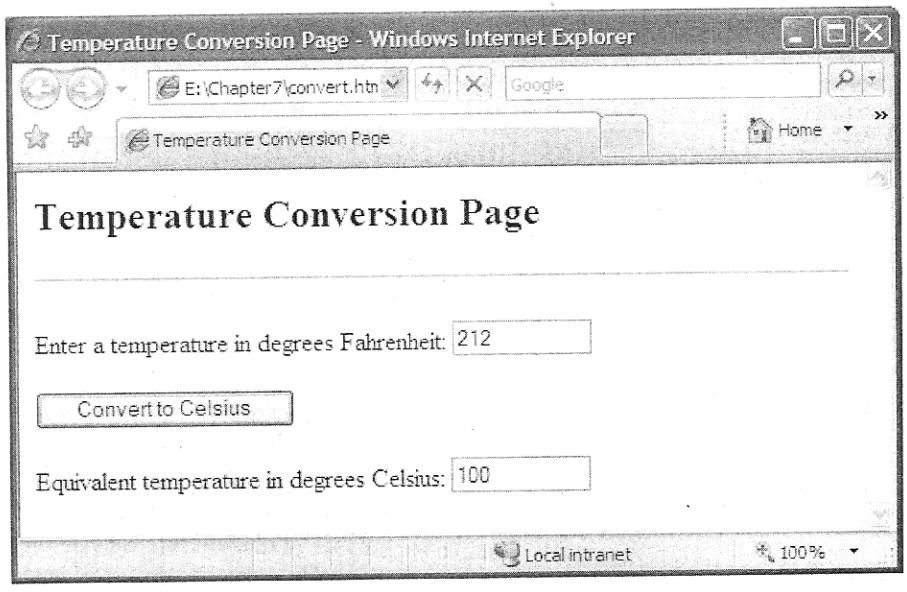


Figure 7.8 `convert.html` rendered in a Web browser.

user input. The user enters a Fahrenheit temperature in `fahrBox` and then clicks the button labeled `"Convert to Celsius"`. When the button is clicked, the code defined in its ONCLICK attribute acesses the temperature value that the user entered in `fahrBox` and computes the corresponding Celsius temperature. JavaScript statements specified by the ONCLICK attribute access the temperature value that the user entered in `fahrBox`, compute the corresponding Celsius temperature, and assign that temperature to a text box named `celsiusBox`. As a result, the converted temperature is displayed in the page, as shown in Figure 7.8

EXERCISE 7.3

Enter the `convert.html` text from Figure 7.7 into a new Web page, then load the page in the browser to verify that it behaves as described.

Add a second button to the page so that the opposite conversion, from Celsius to Fahrenheit, can also be performed. Your revised page should look similar to the one depicted in Figure 7.9. If the user enters a temperature in the Fahrenheit box and clicks the button labeled `"Fahr → Celsius"`, then

Common errors to avoid...

Note that, in the example from Figure 7.7, we called the `parseFloat` function before performing a computation on the text box's contents. Like input returned by the `prompt` function, a value read in from a text box is initially represented as a string. Therefore, you must use `parseFloat` to explicitly convert text-box values before treating them as numbers.

In this example, forgetting to `parseFloat` the value from the text box would not affect the behavior of the page, because applying the subtraction operator would automatically convert the text-box value to a number. However, in cases where a text-box value is added to another value, the browser would interpret the `'+'` operator as specifying concatenation involving the string value. To avoid problems such as this, you should always call `parseFloat` whenever you plan to treat the value in a text box as a number.

Figure 7.9 Sample appearance of temperature conversion page.

the corresponding temperature should appear in the Celsius box. Likewise, if the user enters a temperature in the Celsius box and clicks the button labeled "Fahr ← Celsius", the corresponding temperature should appear in the Fahrenheit box. Recall from Exercise 5.5 that the Celsius can be converted to Fahrenheit using the following formula:

```
tempInFahr=((9/5)*tempInCelsius) + 32;
```

EXERCISE 7.4

Reimplement your `grades.html` page from Chapter 5 (Exercise 5.6) using text boxes and a button. Your new page should include text boxes in which the user can enter each individual grade (homework average, lab average, midterm, and final exam), as well as a single button to compute the course average. When the user clicks the button, the page should determine the average and display it in another text box.

Hint: This page should look a lot like the `convert.html` page from Figure 7.7, except that `grades.html` will contain four text boxes for inputs. Instead of converting a specified temperature from Fahrenheit to Celsius, your button will trigger the calculation of a weighted average and display that average in a separate text box.

Designer Secrets

It is important to recognize that the programming skills you learned in previous chapters are applicable to developing event-driven pages. Most of the pages you wrote involved the same basic steps: (1) obtain inputs from the user *via prompts*, (2) perform some computation on those inputs, then (3) display the results *using write statements*. You placed the code for carrying out these steps within SCRIPT tags in the body of the page, and the browser executed the code when the page loaded.

In an event-driven page, the interface between the user and the code is different—the user enters inputs directly into text boxes, and code executes when the user clicks a button. However, the underlying algorithm for solving problems remains the same: (1) obtain inputs from the user *by accessing text boxes*, (2) perform some computation on those inputs, then (3) display the results *by assigning to text boxes*.

If you compare the code from Chapter 5's `ftoc.html` page (Figure 5.2) with the event-driven version in Figure 7.7, you will see that the two pages are very similar. Although the event-driven page uses text boxes to handle input and output, the statements triggered by the button perform the same tasks that the code from the BODY of the original page does.

Text Boxes for Handling Both Input and Output

It is not uncommon for a particular sequence of JavaScript statements to employ the same variable in multiple ways. For example, a program might use a variable initially to store a user input, then later assign the result of some computation to that same variable. If we extend our analogy of text boxes as similar to variables, it follows that the same text box can be used both to obtain user input and to display the result of a computation.

Consider, for example, the button in Figure 7.10. The statements in the ONCLICK attribute access the number in the text box named `numberBox` multiply that number by two, and assign the result back to the box. Thus, every time the button is clicked, the current value of the text box will be doubled. A user can enter any number in the text box, then click the button repeatedly to update the value displayed in the box.

EXERCISE 7.5

Create a Web page named `double.html` that contains the button from Figure 7.10, as well as a text box named `numberBox` that has an initial value of 1.

If you start with a value of 1 in the text box, how many times must you click the button before the displayed value exceeds 500? How many times to exceed 1,000?

Input/Output via Text Areas

Although text boxes provide a convenient method of reading in and displaying values in Web pages, the fact that text boxes can contain only one line of text often limits their usefulness. An alternative is a ***text area***, which is similar to a text box but can contain any number of text lines. The following represents a generalized text area element:

```
<textarea id="TEXTAREA_ID" rows="NUM_ROWS" cols= "NUM_COLS">
INITIAL_TEXT
</textarea>
```

A text area's ID attribute specifies the identifier for that area, similar to the ID attribute of a text box. The numbers of rows (horizontal lines that can contain text) and of columns (maximum characters per line) in the text area are defined by the attributes ROWS and COLS, respectively. The initial text that appears in the text area (if any) is enclosed between the `<textarea>` and `</textarea>` tags.

The Web page in Figure 7.11 contains two text boxes, each 15 characters wide, and a text area consisting of four rows of 40 characters. The page asks users to enter their first and last names in the text boxes. Once a user enters the names and clicks the button, the program accesses the contents of the text boxes and incorporates the names in a greeting, which displays in the text area. Note that there is no text between the opening and closing TEXTAREA tags (line 28), so the text area initially appears empty. The rendered page is shown in Figure 7.12.

```
<input type="button" value="Double It"
       onclick="num=document.getElementById('numberBox').value;
                num=parseFloat(num);
                document.getElementById('numberBox').value = 2 * num;" />
```

Figure 7.10 JavaScript code that doubles the contents of a text box.

```
<html>
  <!-- greetbox.html                                          Dave Reed -->
  <!-- This page displays a personalized greeting in a text area. -->
  <!-- ========================================================== -->

  <head>
    <title> Long Greeting </title>
  </head>

  <body>
    <p>
      Enter your first name: <input type="text" id="firstNameBox" size="15" />
      <br />
      Enter your last name: <input type="text" id="lastNameBox" size="15" />
    </p>
    <p>
      <input type="button" value="Click for Greeting"
             onclick="firstName = document.getElementById('firstNameBox').value;
                      lastName = document.getElementById('lastNameBox').value;
                      message = 'Hello ' + firstName + ' ' + lastName +
                                ', or may I just call you ' + firstName +
                                '? You wouldn\'t be related to the ' + lastName +
                                's of Park Avenue, would you?';
                      document.getElementById('messageArea').value = message;" />

    </p>
    <p>
      <textarea id="messageArea" rows="4" cols="40"></textarea>
    </p>
  </body>
</html>
```

Figure 7.11 Web page that uses a text area for output.

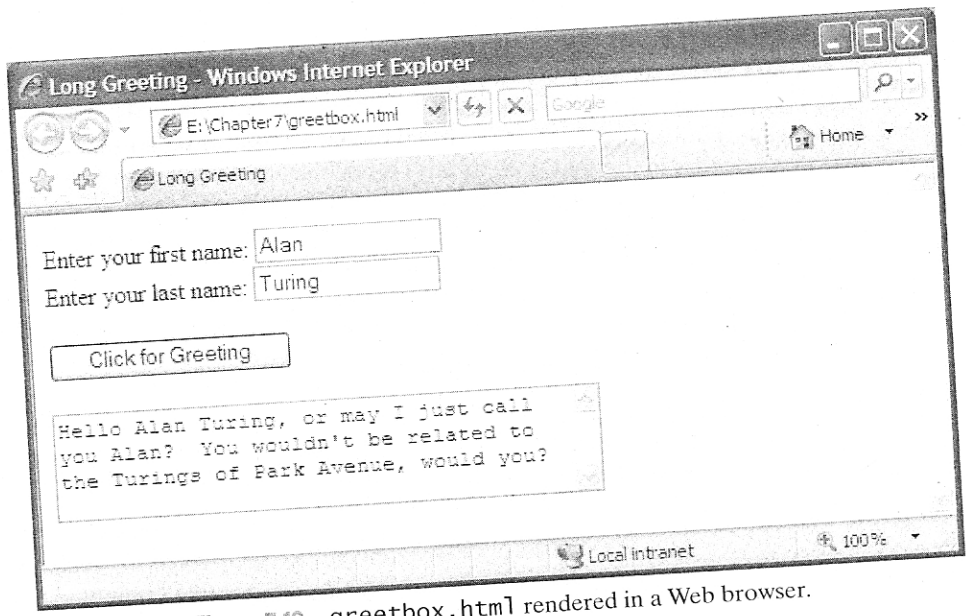


Figure 7.12 `greetbox.html` rendered in a Web browser.

Common errors to avoid...

As we saw in Chapter 2, browsers interpret HTML tags as a page loads, displaying the appropriate formatting and/or elements in the rendered page. Because the contents of a text box or text area can be updated repeatedly after a page loads, the browser does not process text contained in these elements. Therefore, HTML tags cannot be used to format text in a text box or text area. If you mistakenly include HTML tags in the contents of a text box or a text area, the tags will be treated like any other sequence of characters and will be displayed exactly as typed. If you enter extra spaces in the text, they also will appear exactly as you typed them.

If you want to force a line break within a text area, you can include the character sequence `\n` within the text. Inserting this special sequence causes subsequent text to begin on a new line, producing the same effect that `<br />` does in HTML. Similarly, you can include the character sequence `\t` for a tab.

EXERCISE 7.6

Enter the `greetbox.html` text from Figure 7.11 into a new Web page, then load the page in the browser to verify that it behaves as described.

Once you have done this, modify the page so that it acts as an event-driven version of your `story.html` page from Chapter 4. The page should incorporate text boxes in which the user can enter the story's missing words, and each text box should be accompanied by a label (such as `color:` or `noun:`) to identify the type of word expected. Include a button that, when clicked, incorporates the user's words into the story and displays the story in the text area.

Dynamic Images

In Chapter 2, you learned how to insert images in Web pages using the IMG element. For example, suppose that `happy.gif` is an image file containing a picture of a happy face. The following tag, when embedded in a Web page, would cause the `happy.gif` image to appear in the page:

```
<img id="faceImg" src="happy.gif" alt="Happy Face" />
```

This chapter has already demonstrated that user-initiated events can cause the contents of text boxes and text areas to change. However, you can also use events to access and modify IMG elements within a Web page. This ability enables you to design a page that displays different images when the user clicks a button or enters a value in a text box. In such cases, the event triggers JavaScript statements that assign a new file to the IMG element's SRC attribute. For example, if a Web page contained the IMG tag listed above, the following assignment would replace `happy.gif` with an image stored in the file `sad.gif`:

```
document.getElementById('faceImg').src = "sad.gif";
```

Note that this assignment to the IMG element is similar to earlier assignments to text boxes and text areas. However, images do not contain values in the way that text boxes and text areas do. Instead, an IMG element is associated with a source file, so the assignment is made to the SRC attribute of the image (as opposed to a VALUE attribute of a text box or text area).

The Web page in Figure 7.13 contains a single IMG element, which is initially assigned the source file `happy.gif` (assumed to reside in the same directory as the page). When the user clicks the button labeled `"I feel sad"`, the source file assigned to the image is changed to `sad.gif`, which displays a sad face. Likewise, when the user clicks the button labeled `"I feel happy"`, the source is reset back to `happy.gif`, which displays a happy face. See Figure 7.14 for the rendered page.